Subtree Isomorphism is in Random NC

Phillip B. Gibbons
University of California, Berkeley

Richard M. Karp
University of California, Berkeley

Gary L. Miller
University of Southern California

Danny Soroker
University of California, Berkeley

Abstract

Given two trees, a guest tree G and a host tree H, the subtree isomorphism problem is to determine whether there is a subgraph of H that is isomorphic to G. We present a randomized parallel algorithm for finding such an isomorphism, if it exists. The algorithm runs in time $O(\log^3 n)$ on a CREW PRAM, where n is the number of nodes in H. Randomization is used (solely) to solve each of a series of bipartite matching problems during the course of the algorithm. We demonstrate the close connection between the two problems by presenting a log space reduction from bipartite perfect matching to subtree isomorphism. Finally, we present some techniques to reduce the number of processors used by the algorithm.

1. Introduction

A subtree of a tree T is any subgraph of T that is a tree. Given two (unrooted) trees, a guest tree G and a host tree H, the subtree isomorphism problem is to determine whether there is a subtree of H that is isomorphic to G. There is an $O(n^{2.5})$ sequential algorithm for this problem due to Matula[5], where n is the number of nodes in H. In this paper, we present an $O(\log^3 n)$ time randomized algorithm for a CREW PRAM that exhibits the mapping between the trees, if such a mapping exists. We assume the word size of the PRAM is $c \log n$ for some constant c. With a few techniques to reduce the processor count, the algorithm uses $< \sqrt{n} \cdot P(n)$ processors, where $P(n)$ is the number of processors needed for *one* bipartite matching problem on n nodes using the fastest algorithm for bipartite matching to date[7], for a total of $o(n^{4.9})$ processors. More precisely, let $M(n)$ be the number of bit operations required by a CREW PRAM to multiply two $n \times n$ boolean matrices in $O(\log n)$ time. Then our algorithm uses $n^{2.5}M(n)/\log^3 n$ processors.

Our algorithm is based on Matula's sequential algorithm. The main obstacle to developing a fast parallel algorithm from this sequential algorithm is that its running time is proportional to the height of the tree. But by adapting the dynamic tree contraction algorithm of Miller and Reif[6], we show that subtree isomorphism is in random-NC (RNC). Dynamic tree contraction is one of two classic methods for achieving NC and RNC algorithms for problems involving potentially unbalanced trees; the other is recursively finding a vertex "$1/3 - 2/3$" separator for the tree[2]. In a complementary effort, Lingas and Karpinski[4] independently developed an RNC^3 algorithm for subtree isomorphism based on the latter method. In both algorithms, the randomization is used to perform the bipartite matching problems: if a (deterministic) NC algorithm is found for bipartite matching, then subtree isomorphism will also be in NC.

As in Matula's algorithm, we recast subtree isomorphism as a problem on limbs of G and H. A limb of a tree T is a subgraph of T rooted at a vertex u consisting of an edge $\{u, v\}$ of T, together with the connected component of $T - \{\{u, v\}\}$ which contains v. Let $T\langle u, v\rangle$ denote the limb of T defined by the root vertex u and the edge $\{u, v\}$. Any edge $\{v, w\} \in T, w \neq u$, will determine a limb $T\langle v, w\rangle$, a subgraph of $T\langle u, v\rangle$, which we call a child limb of $T\langle u, v\rangle$. If $\{t_1, t_2\}$ is the only edge incident to t_1 in T, then the limb $T\langle t_1, t_2\rangle$ contains all of T, and is denoted a root limb, and $T\langle t_2, t_1\rangle$ is denoted a leaf limb. We say $S\langle s_1, s_2\rangle$ is limb imbeddable in $T\langle t_1, t_2\rangle$ if and only if $S\langle s_1, s_2\rangle$ is isomorphic to a subtree of $T\langle t_1, t_2\rangle$ such that s_1 is mapped to t_1 and s_2 is mapped to t_2. The height of a limb $T\langle t_i, t_j\rangle$ denotes the maximum distance of any vertex of $T\langle t_i, t_j\rangle$ from the root vertex t_i. A unary chain in a tree T is a maximal sequence of edges $\{t_0, t_1\}, \{t_1, t_2\}, \ldots, \{t_{k-1}, t_k\}$, where $T\langle t_i, t_{i+1}\rangle$ is the only child limb of $T\langle t_{i-1}, t_i\rangle$, for $1 \leq i \leq k$, and $T\langle t_k, t_{k+1}\rangle$ is not a leaf limb.

Given two trees G and H, one can test whether there is subtree of H that is isomorphic to G as follows. First choose a root limb $G\langle g_1, g_2\rangle$ of G. G can be imbedded in H if and only if $G\langle g_1, g_2\rangle$ can be limb imbedded in some limb of H (typically not a root limb). To determine whether $G\langle g_i, g_j\rangle$ can be limb imbedded in $H\langle h_k, h_l\rangle$, one can apply the following theorem due to Matula:

Theorem 1 *[5]: Let $G\langle g_i, g_j\rangle$ be a limb of a tree G and $H\langle h_l, h_m\rangle$ be a limb of a tree H. Let A be a bipartite graph, where the boys are the child limbs of $G\langle g_i, g_j\rangle$ and the girls are the child limbs of $H\langle h_l, h_m\rangle$. There is an edge between boy $G\langle g_j, g_k\rangle$ and girl $H\langle h_m, h_n\rangle$ if and only if $G\langle g_j, g_k\rangle$ is limb imbeddable in $H\langle h_m, h_n\rangle$. Then there exists a matching in A that matches all the boys if and only if $G\langle g_i, g_j\rangle$ is limb imbeddable in $H\langle h_l, h_m\rangle$.*

In this way, one can solve an instance of the subtree isomorphism problem by starting at limbs of height 1 and progressing up to the root limb $G\langle g_1, g_2\rangle$, at each height solving a series of bipartite matching problems in order to determine which limb imbeddings are possible.

In order to achieve polylogarithmic running time we will apply dynamic tree contraction[6] to the guest tree. At each phase of the contraction process the following two operations are applied to the current tree $\hat{G}$: the rake operation, which deletes the edges incident with the leaves of $\hat{G}$, and the compress operation, which pairs up edges along unary chains in $\hat{G}$ and replaces each pair by a single edge. As the contraction process proceeds, information about the possible limb imbeddings of G in H is accrued. In particular, for each limb in $\hat{G}$ with exactly one child limb, we compute and maintain a set of conditionals defining its limb imbeddability in H as a function of the as-yet-unknown limb imbeddability of its one child limb. As part of a compress operation, such conditionals for a limb can be readily composed with the conditionals for its child. By the time $\hat{G}$ is a single edge, sufficient information is available to determine whether the root limb of G is limb imbeddable in any limb of H. If so, an explicit isomorphism of G to a subtree of H is constructed during a second iterative process which expands G back to its original size by reversing the contraction process.

To reduce the number of processors, we will apply two bipartite matching algorithms: one for the *decision* problem (determine if a perfect matching exists) while contracting the tree, and one for the *search* problem (find the edges in a perfect matching) while expanding the tree. Further processor savings are achieved by solving groups of related matching problems at once. Finally, we show that bipartite perfect matching and subtree isomorphism are mutually NC reducible by presenting a log space reduction from the former to the latter.

Miller and Reif[6] use dynamic tree contraction to develop NC algorithms for the related problems of tree isomorphism, canonical labels for trees, and canonical labels for all subtrees. The latter problem assigns labels to all nodes in a rooted tree such that two nodes u and v have the same label if and only if the *maximal* subtree rooted at u is isomorphic to the *maximal* subtree rooted at v. This differs from the subtree isomorphism problem, in which the subtrees of H are not necessarily maximal.

2. The subtree isomorphism algorithm

Let n_G be the number of nodes in G and $n = n_H$ be the number of nodes in H. Let G be rooted at a root limb $G\langle g_1, g_2\rangle$. This determines a set of $n_G - 1$ limbs associated with the rooted G. As the algorithm proceeds, it contracts G using suitably defined rake and compress operations. Throughout the algorithm, we will associate the limb $\hat{G}\langle g_i, g_j\rangle$ with its second vertex g_j, and name the limb $\hat{G}\langle \star, g_j\rangle$ to reflect the fact that the parent node of g_j may change as a result of compress operations. The host tree H is not rooted and thus has $2(n_H - 1)$ limbs to consider. As the algorithm does not alter H, we will name its limbs with two vertices, e.g. $H\langle h_k, h_l\rangle$.

As the contraction process proceeds, the algorithm maintains a correspondence between the limbs that remain in the contracted tree $\hat{G}$ and the limbs of the original rooted tree G. The limb $G\langle g_p, g_q\rangle$ of G corresponding to the limb $\hat{G}\langle \star, g_j\rangle$ of $\hat{G}$ is called the original limb of $\hat{G}\langle \star, g_j\rangle$. At the beginning of the process, each limb of G is its own original limb. The assignment of original limbs to current limbs changes whenever a compress operation takes place. Consider the case where edges $\{g_i, g_j\}$ and $\{g_j, g_k\}$ on a unary chain are compressed into a single edge $\{g_i, g_k\}$. Suppose that, before the compress operation, the original limb of $\hat{G}\langle \star, g_j\rangle$ is $G\langle g_p, g_q\rangle$. Then, after the compress operation, the limb $\hat{G}\langle \star, g_j\rangle$ is no longer present, and the original limb of $\hat{G}\langle \star, g_k\rangle$ becomes $G\langle g_p, g_q\rangle$. Based on this correspondence, we say a limb $\hat{G}\langle \star, g_j\rangle$ in $\hat{G}$ is limb imbeddable in $H\langle h_k, h_l'\rangle$ if and only if the original limb of $\hat{G}\langle \star, g_j\rangle$ is limb imbeddable in $H\langle h_k, h_l\rangle$.

At the conclusion of each contract phase, the following data will have been computed:

- for each leaf limb $\hat{G}\langle\star, g_j\rangle$ currently in $\hat{G}$ or removed at an earlier rake step, a value giving the set of H limbs in which $\hat{G}\langle\star, g_j\rangle$ can be limb imbedded;
- for each limb $\hat{G}\langle\star, g_i\rangle$ with exactly one child limb, a partial value consisting of a collection of conditional sets.

The conditional sets are defined as follows: let $\hat{G}\langle\star, g_j\rangle$ be the child limb of $\hat{G}\langle\star, g_i\rangle$. Then the conditional set of the pair $\hat{G}\langle\star, g_i\rangle$, $H\langle h_k, h_l\rangle$ contains the limb $H\langle h_m, h_n\rangle$ if and only if determining that $\hat{G}\langle\star, g_j\rangle$ is limb imbeddable in $H\langle h_m, h_n\rangle$ would yield the fact that $\hat{G}\langle\star, g_i\rangle$ is limb imbeddable in $H\langle h_k, h_l\rangle$. A short hand notation for this situation is "$H\langle h_k, h_l\rangle$ if $H\langle h_m, h_n\rangle$". The partial value of a limb with exactly one child limb is the collection of its $2(n_H - 1)$ conditional sets.

2.1. Pseudo-code for the algorithm

Our algorithm uses the following data structures. Let **Imbed**[,] be an $(n_G - 1) \times (2n_H - 2)$ matrix, with one entry for each pair $\hat{G}\langle\star, g_i\rangle$,$H\langle h_j, h_k\rangle$. During the course of the algorithm, **Imbed**$[g_i, \langle h_j, h_k\rangle]$ will be set to 1 if and only if $\hat{G}\langle\star, g_i\rangle$ is found to be limb imbeddable in $H\langle h_j, h_k\rangle$. If it is set to 1, then $\hat{G}\langle\star, g_i\rangle$ must be a leaf. Thus the original limb of $\hat{G}\langle\star, g_i\rangle$ will no longer change, and **Imbed**$[g_i, \langle h_j, h_k\rangle]$ will be valid for the rest of the algorithm. Let **Conditionals**[,,] be an auxiliary $(n_G - 1) \times (2n_H - 2) \times (2n_H - 2)$ matrix, used for storing the conditional sets of all pairs of limbs. **Conditionals**$[g_i, \langle h_j, h_k\rangle, \langle h_l, h_m\rangle]$ will be set to 1 if and only if $H\langle h_l, h_m\rangle$ is in the conditional set of the pair $\hat{G}\langle\star, g_i\rangle$,$H\langle h_j, h_k\rangle$.

Algorithm C below gives a pseudo-code description of our subtree isomorphism algorithm.

Theorem 2 *Given two trees G and H, algorithm C determines if there is a subtree of H isomorphic to G.*

Proof: We omit the proof. It will appear in the full paper. □

2.2. Analysis of the algorithm

The resource bounds for our algorithm are bounded by the time and processor count needed for steps C9 and C13. We will use the Mulmuley, Vazirani, Vazirani randomized algorithm[7] for constructing a perfect matching in a bipartite graph. This algorithm takes $O(\log^2 n)$ time and $n^2 M(n) \log\log n/\log n$ processors, i.e. $o(n^{4.4})$ processors, where n is the number of nodes in each half of the bipartite graph.

For step C9, i.e. steps C19–C22, for each g_i, $O(n_H)$ bipartite matching problems of size $\leq n_H$ are solved in parallel. For each g_i, this requires $O(\log^2 n_H)$ time and $n_H^3 M(n_H) \log\log n_H/\log n_H$ processors. Similarly, for step C13, i.e. steps C23–C27, for each g_i, $O(n_H^2)$ bipartite matching problems of size $\leq n_H$ are solved. This requires $O(\log^2 n_H)$ time and $n_H^4 M(n_H) \log\log n_H/\log n_H$ processors for each g_i.

Algorithm C: Given a tree H on n_H vertices $\{h_1, \ldots, h_{n_H}\}$ and a tree G on n_G vertices $\{g_1, \ldots, g_{n_G}\}$, this algorithm determines if there is a subtree of H isomorphic to G.

```
1.   Select a root limb $G\langle g_1, g_2\rangle$ of $G$, and for all $g_i$ except $g_1$,
         let $g_i.parent$ be the parent of $g_i$ in the resulting rooted tree $\hat{G}$.

2.   Initialize the Imbed and Conditionals matrices to all zeroes. Then for each
         leaf limb $\hat{G}\langle\star, g_i\rangle$, set Imbed$[g_i, \langle h_j, h_k\rangle]$ to 1 for all $H$ limbs $H\langle h_j, h_k\rangle$.
         For each limb $\hat{G}\langle\star, g_i\rangle$ with exactly one child limb: for all $H$ limbs $H\langle h_j, h_k\rangle$,
         set Conditionals$[g_i, \langle h_j, h_k\rangle, \langle h_k, h_l\rangle]$ to 1 for all its child limbs $H\langle h_k, h_l\rangle$.

3.   WHILE there exists $>$ 1 edges in $\hat{G}$ DO:
4.       IN PARALLEL for each limb $\hat{G}\langle\star, g_i\rangle$ in $\hat{G}$ DO:

             /* rake all leaves, update their parents accordingly */
5.           IF leaf limb
6.               mark $g_i$ as deleted from $\hat{G}$

7.           ELSE IF all its child limbs are leaf limbs
8.               IF $>$ 1 child limbs
9.                   Find_Imbeddings_For_Limb($g_i$)

                 ELSE          /* exactly 1 child limb */
10.                  determine the set of $H$ limbs in which $\hat{G}\langle\star, g_i\rangle$ is limb imbeddable
                         from its conditionals and the $H$ limbs in which its remaining
                         child limb $\hat{G}\langle\star, g_j\rangle$ is now known to be limb imbeddable,
                         e.g. if Conditionals$[g_i, \langle h_k, h_l\rangle, \langle h_m, h_n\rangle] = 1$ and
                         Imbed$[g_j, \langle h_m, h_n\rangle] = 1$, set Imbed$[g_i, \langle h_k, h_l\rangle]$ to 1.

11.          ELSE IF some of its child limbs are leaf limbs and some are not
12.              IF exactly 1 nonleaf child limb $\hat{G}\langle\star, g_j\rangle$
13.                  Find_Conditional_Imbeddings_For_Limb($g_i, g_j$)

             /* compress all unary chains */
             ELSE          /* none of its child limbs are leaf limbs */
14.              IF exactly 1 child limb AND edge $\{g_i$.parent$, g_i\}$
                         is of even parity on its unary chain
15.                  compose the conditionals associated with $g_i$ and $g_i$.parent,
                         e.g. "$H\langle h_i, h_j\rangle$ if $H\langle h_k, h_l\rangle$" in parent and "$H\langle h_k, h_l\rangle$
                         if $H\langle h_m, h_n\rangle$" in $g_i$ results in "$H\langle h_i, h_j\rangle$ if $H\langle h_m, h_n\rangle$",
                         marking Conditionals accordingly.

16.                  mark $g_i$.parent as deleted from $\hat{G}$
17.                  $g_i$.parent $\leftarrow$ ($g_i$.parent).parent

18.  Let $\hat{G}\langle\star, g_i\rangle$ be the remaining limb in $\hat{G}$. If there is an $H\langle h_k, h_l\rangle$ such that
         Imbed$[g_i, \langle h_k, h_l\rangle] = 1$, then there is a subtree of $H$ isomorphic to $G$.
```

PROCEDURE Find Imbeddings For Limb(g'):

```
19.  IN PARALLEL for each limb $H\langle h,h'\rangle$ of $H$ DO:
20.      IF $H\langle h,h'\rangle$ has at least as many child limbs as $G\langle g'.\text{parent}, g'\rangle$
21.          Set up a bipartite matching problem $P$ where the boys are the child
                 (leaf) limbs $\dot{G}\langle\star,g_1'\rangle,\ldots,\dot{G}\langle\star,g_k'\rangle$ of $\dot{G}\langle\star,g'\rangle$ currently in $\hat{G}$ or
                 removed by an earlier rake operation, and the girls are the child
                 limbs $H\langle h',h_1'\rangle,\ldots,H\langle h',h_l'\rangle$ of $H\langle h,h'\rangle$. There is an edge between
                 boy $\dot{G}\langle\star,g_i'\rangle$ and girl $H\langle h',h_j'\rangle$ if and only if $\dot{G}\langle\star,g_i'\rangle$ can be
                 limb imbedded in $H\langle h',h_j'\rangle$, i.e. Imbed$[g_i',\langle h',h_j'\rangle] = 1$. Also,
                 add dummy limbs (with edges to all the girls) to the set
                 of boys, to make the number of boys equal the number of girls.

22.          Find a perfect matching in $P$. If one exists, set Imbed$[g',\langle h,h'\rangle]$ to 1.
```

PROCEDURE Find Conditional Imbeddings For Limb(g', g_x'):

```
23.  IN PARALLEL for each limb $H\langle h,h'\rangle$ of $H$ DO:
24.      IF $H\langle h,h'\rangle$ has at least as many child limbs as $G\langle g'.\text{parent}, g'\rangle$
25.          IN PARALLEL for all child limbs $H\langle h',h_j'\rangle$ of $H\langle h,h'\rangle$ DO:
26.              Set up a bipartite matching problem $P'$ as in C21, except exclude
                     child limbs $\dot{G}\langle\star,g_x'\rangle$ and $H\langle h',h_j'\rangle$ from the matching problem.

27.              Find a perfect matching in $P'$. If one exists,
                     set Conditionals$[g',\langle h,h'\rangle,\langle h',h_j'\rangle]$ to 1.
```

Lemma 1 *Let G be a tree rooted at a root limb. Let $G\langle g_i, g_j\rangle$ be a limb in G that has c child limbs. Let the contraction process (as defined in Algorithm C) be applied to G until the tree is contracted to 1 edge. Then (1) there are no bipartite matching problems solved for node g_j if $c \le 1$, (2) there is exactly one phase in which there are bipartite matching problems solved for node g_j if $c > 1$, and (3) prior to the matching problems during this phase, $G\langle g_i, g_j\rangle$ will be the original limb of $\hat{G}\langle\star, g_j\rangle$.*

Proof: The main observation behind the proof is that we perform a *lazy* rake operation, one that solves matching problems at a node g_j only when the rake operation will result in $\hat{G}\langle\star, g_j\rangle$ having 0 or 1 child limbs. □

$O(\log n_G)$ iterations of the WHILE loop suffice to contract G to 1 edge[6], and so steps C3–C17 will take $O(\log n_G \log^2 n_H)$ time. By lemma 1, step C9 or step C13 will be executed at most once for each g_i. It follows that algorithm C runs in $O(\log n_G \log^2 n_H)$ time on a CREW PRAM with $o(n_G n_H^{6.4})$ processors. In the next section, we show how the processor count can be significantly reduced.

In order to exhibit an explicit isomorphism of G to a subtree of H, we make the following additions to algorithm C. While contracting the tree, count the number of contract phases applied so far, in order to save the "time" each node was deleted. When a node is deleted as a result of a rake operation, also save the name of its parent; for a compress operation, save the name of its child. Save all perfect matchings constructed.

After G has been contracted, we reconstruct G by an expansion process which reverses the contraction process, with each expand phase splicing back into $\hat{G}$ all nodes and edges deleted at the corresponding contract phase. At the conclusion of each expand phase, we will have computed the home limb for each limb in the current $\hat{G}$. The home limb of a limb $\hat{G}\langle\star, g_i\rangle$ is the limb $H\langle h_k, h_l\rangle$ such that the isomorphism being constructed maps the (current) original limb of $\hat{G}\langle\star, g_i\rangle$ to $H\langle h_k, h_l\rangle$. Because they are associated with the original limbs of the current $\hat{G}$ limbs, these home limbs typically will be scattered throughout H prior to the final expand phase. During the expansion process, new home limbs are computed based on the matchings performed during the contraction process and the home limbs of existing limbs in $\hat{G}$.

Clearly the time and processor count for exhibiting the imbedding is bounded by the time and processor count for algorithm C.

3. Processor efficiency

We have recast the subtree isomorphism problem as a problem on limbs, as in Matula's algorithm, in order to save having to try out all possible roots for the trees. In this section, we show how to reduce further the number of processors needed. First, use a *decision* algorithm for steps C22 and C27. In addition, while contracting the tree, save the "time" the matching problems were solved for each node. Then, construct the necessary matchings while expanding the tree. We expand the tree as described in section 2, with the following modification. At the beginning of each expand phase, if (decision) bipartite matching problems were solved for node g_i at the corresponding contract phase, construct the matching and save the results. There are two cases to consider. (1) If step C9 was performed, then the home limb $H\langle h_j, h_k\rangle$ of $\hat{G}\langle\star, g_i\rangle$ is known, so it suffices to solve only one (search) bipartite matching problem for g_i: the matching problem for the pair $\hat{G}\langle\star, g_i\rangle$, $H\langle h_j, h_k\rangle$. (2) If step C13 was performed, then both the home limb $H\langle h_k, h_l\rangle$ of $\hat{G}\langle\star, g_i\rangle$ and the home limb $H\langle h_l, h_m\rangle$ of the remaining child limb $\hat{G}\langle\star, g_j\rangle$ of $\hat{G}\langle\star, g_i\rangle$ are known, so it suffices to solve only one (search) bipartite matching problem for the pair $\hat{G}\langle\star, g_i\rangle$, $H\langle h_k, h_l\rangle$ where $\hat{G}\langle\star, g_j\rangle$ and $H\langle h_l, h_m\rangle$ are excluded from the matching problem.

Lemma 2 *During the expansion process, there is at most one bipartite matching problem solved for each node in the rooted H.*

Proof: By lemma 1, there will be at most one bipartite matching problem solved for each node g_j during the expansion process, and this matching problem determines the home limb of $G\langle g_i, g_j\rangle$. □

The running time for expanding the tree is $O(\log n_G \log^2 n_H)$, using the Mulmuley, Vazirani, Vazirani algorithm for constructing perfect matchings. Define the work of an algorithm to be the sum over all processors p_i of the number of PRAM instructions/operations executed by p_i during the course of the algorithm. Clearly the work to expand the tree is dominated by the work to construct the matchings. Let d_j be the degree of node h_j. By lemma 2, the work is $\leq \sum_{j=1}^{n_H} d_j^2 M(d_j) \log d_j \log\log d_j$, i.e. $\in O(n_H^2 M(n_H) \log n_H \log\log n_H)$

We will now analyze the complexity of contracting the tree using a *decision* algorithm for bipartite matching instead of a *search* algorithm. Recall that a bipartite graph has a perfect matching if and only if a certain symbolic matrix is nonsingular[3]. Based on this fact, Borodin, von zur Gathen, Hopcroft[1] developed a randomized algorithm for deciding if a bipartite graph has a perfect matching that runs in $O(\log^2 n)$ time on a CREW PRAM. An improved version of their algorithm computes determinants over Z_p, the integers modulo some suitable prime p of size $O(n^4)$[9]. This can be done with $O(\sqrt{n}M(n))$ work on a CREW PRAM, using the Preparata and Sarwate algorithm[8] for computing the adjoint and the determinant of a matrix, since all operations involve $O(\log n)$-bit numbers.

While contracting the tree, we can save processors by solving groups of related matching problems at once as follows. The a_{ij} entry of the adjoint of a matrix A contains the determinant of the minor A_{ji} (the $(j,i)^{\text{th}}$ *cofactor*). Thus by testing whether a cofactor is 0, we can determine if a perfect matching exists when the parent limb and any one H limb are left out of the matching problem. From Rabin and Vazirani[9], it follows that this holds even when the adjoint is computed over Z_p. Thus, in order to find in which parent limbs $H\langle h_j, h_k\rangle$ adjacent to h_k the limb $\hat{G}\langle\star, g_i\rangle$ can be limb imbedded, we solve one bipartite matching problem where the boys are the child limbs of $\hat{G}\langle\star, g_i\rangle$(currently in $\hat{G}$ or removed by an earlier rake operation), and the girls are the child limbs $H\langle h_k, h_j\rangle$ adjacent to h_k. Add dummy boys to match the number of girls as in step C21, except that one of these dummy boys is designated to correspond to $\hat{G}\langle\star, g_i\rangle$. From the matrix adjoint computed, test the cofactors of the dummy row corresponding to the parent $\hat{G}\langle\star, g_i\rangle$. $\hat{G}\langle\star, g_i\rangle$ is limb imbeddable in $H\langle h_j, h_k\rangle$ if and only if the cofactor in row $\hat{G}\langle\star, g_i\rangle$ and column $H\langle h_j, h_k\rangle$ is $\neq 0$. In this way, for each node g_i, $\leq n_H$ bipartite matching problems (one per each h_j) are solved, each with $\leq d_j = \deg(h_j)$ children. The work to solve these matching problems is $\leq n_G \sum_{i=1}^{n_H} \sqrt{d_i} M(d_i)$, i.e. $\in O(n_G \sqrt{n_H} M(n_H))$ This technique can also be applied to conditional matching problems, where the parent is left out of the graph and the dummy row corresponds to the remaining child (or vice-versa). This results in $\leq 2n_H - 2$ bipartite matching problems solved for each node g_i, each with $\leq n_H$ children.

Let algorithm C' be the improved version of algorithm C, which uses the above steps to save processors and to exhibit the mapping.

Theorem 3 *Given a tree H on n_H vertices and a tree G on n_G vertices, algorithm C' determines, with probability $\geq 1/2$, if there is a subtree of H isomorphic to G, and exhibits the mapping. It runs in time $t \in O(\log n_G \log^2 n_H)$ on a CREW PRAM with $(n_G + n_H^{.5} \log n_H \log\log n_H) n_H^{1.5} M(n_H)/t$ processors, i.e. $O(\log^3 n)$ time with $n^{2.5}M(n)/\log^3 n$ processors.*

4. Reducing bipartite matching to subtree isomorphism

In this section we show that bipartite perfect matching is log space reducible to subtree isomorphism. (Lingas and Karpinski[4] independently discovered an NC^1 reduction of bipartite perfect matching to subtree isomorphism).

Let $A = (X, Y, E)$ be a bipartite graph, where $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. We will construct trees T_x, T_y corresponding to the vertex sets X and Y, such that every imbedding of T_x in T_y yields, in a natural way, a perfect matching in A. It is convenient to view T_x and T_y as rooted at R_x and R_y respectively. This creates no obstacle since our construction forces R_x to be mapped to R_y in any imbedding. The structure of the trees is as follows:

T_x : R_x has $n+2$ children - $X_1, X_2, \ldots, X_n, V_1, V_2$. X_i corresponds to vertex x_i in A. V_1 and V_2 have no children. For $1 \le i \le n$, X_i is the parent of i children, X_{ij}, each of which is a root of a path of length $n - i + 1$.

T_y : R_y has $n+2$ children - $Y_1, Y_2, \ldots, Y_n, U_1, U_2$. Y_i corresponds to vertex y_i in A. V_1 and V_2 have no children. For $1 \le i \le n$, Y_i is the parent of n children, Y_{ij}, where Y_{ij} is the root of a path of length $n - j + 1$ if $\{y_i, x_j\} \in E$ and length $n - j$ otherwise.

Note that this reduction can clearly be performed in log space.

Theorem 4 *T_x is imbeddable in T_y if and only if A has a perfect matching.*

It follows that the problem of *deciding* if a bipartite graph has a perfect matching is log space reducible to the problem of deciding if a tree is isomorphic to a subtree of another tree, and the problem of *constructing* a perfect matching in a bipartite graph is log space reducible to the problem of constructing an imbedding of a tree into another tree. From the reduction, we observe that the number of imbeddings of T_x in T_y is $2n!(n-1)!(n-2)!\cdots 2!$ times the number of perfect matchings of A. Thus the problem of determining the number of imbeddings of a tree in another tree is $\#P$-complete.

5. Remarks

Suppose we use a search algorithm for bipartite matching while contracting the guest tree. Then we can extend our Monte Carlo algorithm to a Las Vegas algorithm as follows. Given a matching M, we can determine whether M is maximum as follows. By directing all matched edges from the boys to the girls, directing all unmatched edges from the girls to the boys, and applying a shortest path calculation in this directed graph, we can find an augmenting path or prove that none exists. The resulting algorithm runs in expected time $O(\log n_G \log^2 n_H)$ with $n_G n_H^2 M(n_H) \log\log n_H / (\log n_G \log n_H)$ processors.

The case where G (or H) has bounded maximum degree d can be done *deterministically* in time $O(d \log^2 n_H \log n_G)$ on a CREW PRAM. Simply solve each bipartite matching problem in our algorithm using d applications of the above method for finding an augmenting path in parallel.

With appropriate implementation, our algorithm is in RNC^3. To see this, observe (1) without the matchings, our algorithm runs in $O(\log^2 n)$ time, and can be implemented on an NC circuit of depth $O(\log^3 n)$, and (2) the bipartite matching algorithms used are in RNC^2.

Acknowledgement

The first author was supported by the International Computer Science Institute (ICSI), Berkeley, CA and by an IBM Doctoral Fellowship. The second author was supported in part by ICSI and NSF Grant #DCR-8411954. The third author was supported in part by NSF Grant #DCR-8514961 and by the Mathematical Sciences Research Institute (MSRI), Berkeley, CA. The fourth author was supported by ICSI and by Defense Advanced Research Projects Agency (DoD) Arpa Order #4871, monitored by Space and Naval Warfare Systems Command under Contract #N00039-84-C-0089. He is currently at IBM Almaden Research.

References

[1] A. Borodin, J. von zur Gathen, and J. Hopcroft. Fast parallel matrix and GCD computations. In *Proc. of the Symp. on Foundations of Computer Science (FOCS)*, Oct. 1982.

[2] R. Brent. The parallel evaluation of general arithmetic expressions. *JACM*, 21:201–208, 1974.

[3] J. Edmonds. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bureau of Standards*, 71B:241–245, 1967.

[4] A. Lingas and M. Karpinski. Subtree isomorphism and bipartite perfect matching are mutually NC reducible. 1987. submitted for publication.

[5] D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. *Annals of Discrete Mathematics*, 2:91–106, 1978.

[6] G. L. Miller and J. H. Reif. Parallel tree contraction and its applications. In *Proc. of the Symp. on Foundations of Computer Science (FOCS)*, Oct. 1985.

[7] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[8] F. Preparata and D. Sarwate. An improved parallel processor bound in fast matrix inversion. *Information Processing Letters*, 7(3):148–150, 1978.

[9] M. O. Rabin and V. V. Vazirani. *Maximum Matchings in General Graphs through Randomization.* Technical Report TR-15-84, Aiken Computation Laboratory, Harvard University, Oct. 1984.